UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA
Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica, Big Data Analytics

Relazione attività Graph Analytics

Studente:

Lorenzo Stigliano
Matricola 185534

Anno Accademico 2022-2023

Contents

1 Introduzione

Il tema della relazione è il caso delle elezioni americane del 2016, all'interno delle quali e' stato fatto un uso massiccio di Twitter per orientare il parare elettorale. In questo scenario e' stato riscontrato come il contributo della Russia sia stato decisivo per poter creare dei flussi di (dis)informazione attraverso Twitter che potessero ostacolare la campagna elettorale di Hillary Clinton. Questo ha creato una vera e propria guerra di disinformazione a colpi di bufale.

Figure 1: Hillary Clinton and Donald Trump

La sanbox utilizzata quindi raccoglie i dati relativi ad utenti (normali o fittizi pilotati dai russi), dei tweet e delle attività legate ad essi durante il periodo delle elezioni.

1.1 Presentazione del Dataset

Il dataset che è utilizzato all'interno della relazione fa parte della collezione di sanbox del sito `https://neo4j.com/`, utilizzabili per studiare le funzionalità offerte da neo4j. Il grafo consiste in un 281136 nodi secondo le seguenti tipologie:

- Tweet
- Hashtag
- URL
- Source
- User
- Troll

Andando a conformarsi come segue:

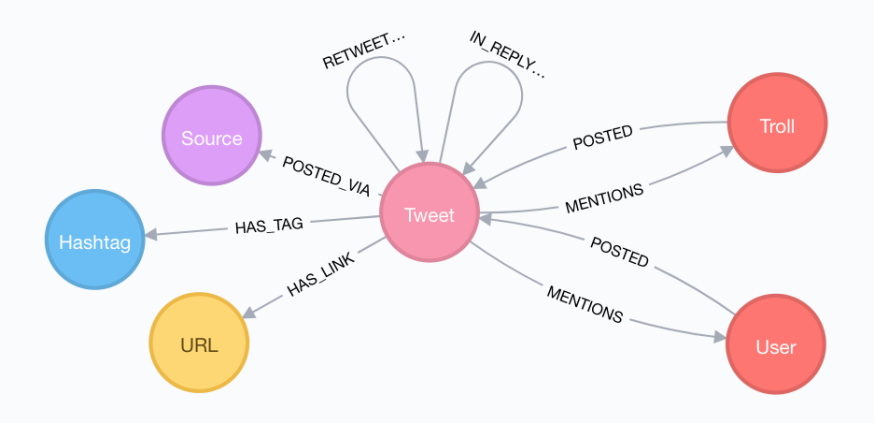


Figure 2: Graph Configuration

Ogni nodo possiede infine i seguenti attributi:

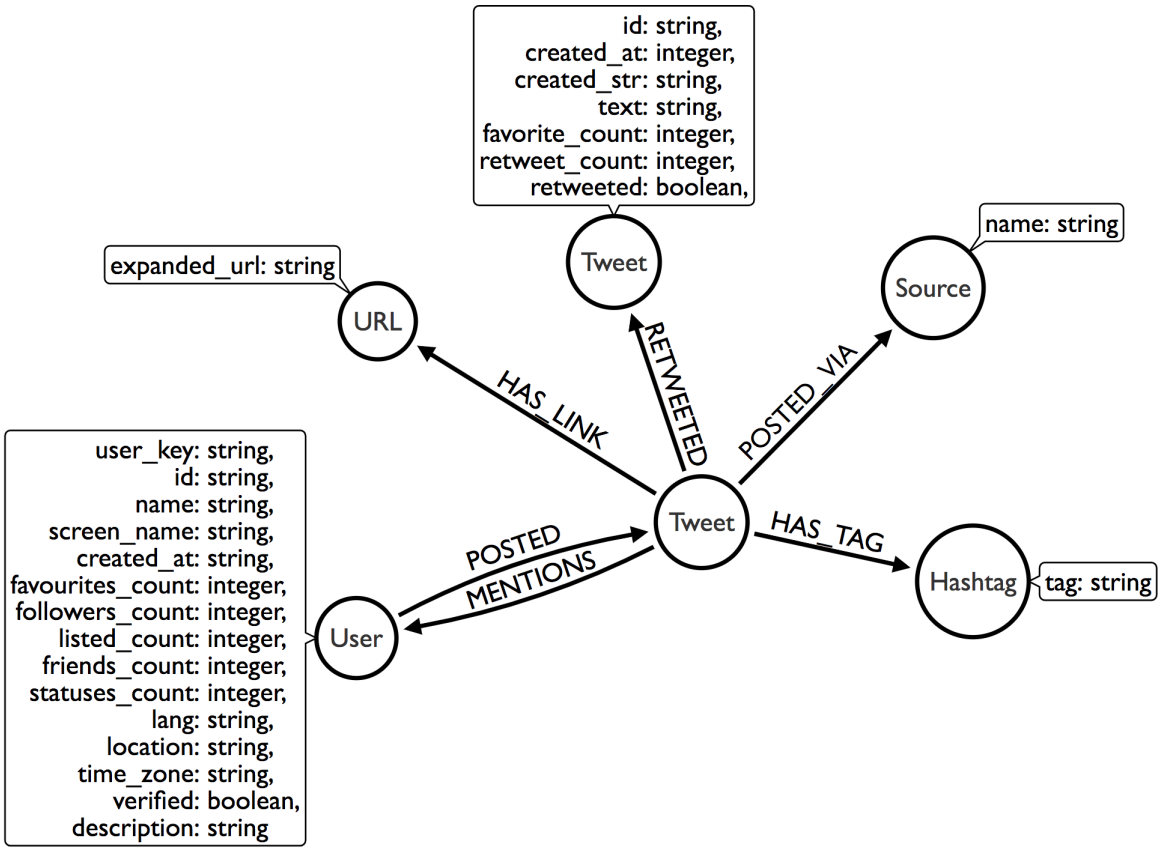


Figure 3: Node Attributes

2 Research Questions

2.0.1 Quali sono gli account più influenti all'interno del dataset?

Per strutturare una ricerca in cui si è interessati a conoscere quali sono gli attori al centro della rete del *social network* si potrebbe immaginare di partire da una `projection` di tipo `cypher` su una porzione di grafo. Si sceglie di eseguire una proiezione poiché in presenza di grandi quantità di dati è un inutile *overhead* dover maneggiare una istanza completa, sapendo a priori che solamente una parte sarà quella effettivamente interessata dall'indagine. Si decide quindi di identificare gli utenti:

- User
- Troll

Ed i *tweets* che hanno: *tweettato*, *re-tweettato* o in cui sono stati menzionati, secondo le seguenti relazioni:

- POSTED
- RETWEETED
- MENTION

Si effettuano innanzitutto effettuati dei match sulla categoria *User* e poi su quella dei *Troll*. Infine, con la direttiva `UNION` vengono uniti i due *match* della proiezione in un solo risultato:

```
CALL gds.graph.project.cypher(
'twitter-user',
'MATCH (u:User) return id(u) as id',
'MATCH (u1:User)-[:POST]->(:Tweet)<-[:RETWEETED]-(:Tweet)<-[:POSTED]-(u2:User)
RETURN id(u2) as source, id(u1) as target
UNION
MATCH (r1:Troll)-[:POSTED]->(:Tweet)<-[:RETWEETED]-(:Tweet)<-[:POSTED]-(r2:Troll)
RETURN id(r2) as source, id(r1) as target
UNION
MATCH (u1:User)<-[:MENTION]-(t:Tweet)<-[:POST]-(u2:User)
RETURN id(u2) as source, id(u1) as target
UNION
MATCH (t1:Troll)<-[:MENTION]-(t:Tweet)<-[:POST]-(t2:Troll)
RETURN id(t2) as source, id(t1) as target')
```

Considerata la porzione di grafo proiettata, si implementa l'algoritmo *Degree Centrality* per andare a individuare quali siano i protagonisti del nostro grafo. Eccone l'implementazione in *neo4j*, in cui vengono visualizzate su due colonne distinte: il nome dell'utente ed il valore assegnatogli dall'algoritmo per determinarne la centralità:

```
CALL gds.degree.stream('twitter-user')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).screen_name AS name, score
ORDER BY score DESC LIMIT 10;
```

L'algoritmo *Degree Centrality* può essere utilizzato per trovare dei nodi importanti all'interno del grafo. Il calcolo viene effettuato analizzando le relazioni, in quanto numero ed orientamento, di ogni singolo nodo.

"name"	"score"
"LazyKStafford"	28.0
"_NickLuna_"	27.0
"MichelleArry"	27.0
"c__wells"	26.0
"JeffreyKahunas"	25.0
"JeanneMccarthy0"	24.0
"DorothieBell"	23.0
"LeroyLovesUSA"	23.0
"GiselleEvns"	23.0
"hollandpatrickk"	22.0

Figure 4: Degree Centrality Result

In *neo4j* è possibile salvare i risultati appena ottenuti in memoria, oppure inserirli all'interno del grafo appena proiettato. Le direttive preposte alle suddette operazioni sono rispettivamente:

- `write`
- `mutate`

Di seguito un esempio dell'implementazione della direttiva `mutate`:

```
CALL gds.degree.mutate('twitter-user', { mutateProperty: 'degree' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean
AS meanScore, nodePropertiesWritten
```

Qualora un utente valutasse fosse incerto rispetto all'utilizzo di risorse di un determinato può fare appello alla direttiva `estimate`:

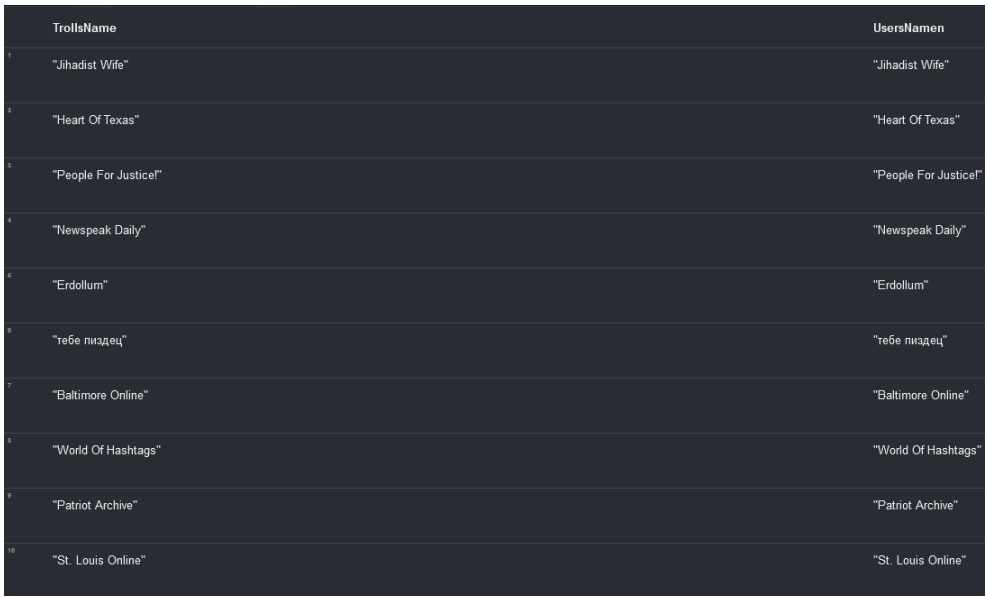
```
CALL gds.degree.mutate.estimate('twitter-user', { mutateProperty: 'degree' })
YIELD requiredMemory, bytesMin, bytesMax,
heapPercentageMin, heapPercentageMax, nodeCount, relationshipCount;
```

"requiredMemory"	"bytesMin"	"bytesMax"	"heapPercentageMin"	"heapPercentageMax"	"nodeCount"	"relationshipCount"
"56 Bytes"	56	56	0.1	0.1	14273	834

Figure 5: Resource Use

Definita quindi la proiezione è possibile interrogare ulteriormente il database, approfondendo con più precisione rispetto agli utenti che sono centrali all'interno del grafo. Per fare questo è stata strutturata una *query* all'interno della quella vengono ricercati gli utenti (*Troll* o *User*) che hanno un numero di *followers* maggiore di 5000 utenti.

```
CALL gds.graph.nodeProperty.stream('twitter-user', 'degree')
YIELD nodeId, propertyValue
WITH gds.util.asNode(nodeId) AS Id, propertyValue AS scId
MATCH (t:Troll)
WHERE t.name = Id.name AND t.followers_count > 5000
MATCH (u:Troll)
WHERE u.name = Id.name AND u.followers_count > 5000
RETURN t.name as TrollsName, u.name as UsersNamen
```



	TrollsName	UsersNamen
1	"Jihadist Wife"	"Jihadist Wife"
2	"Heart Of Texas"	"Heart Of Texas"
3	"People For Justice!"	"People For Justice!"
4	"Newspeak Daily"	"Newspeak Daily"
5	"Erdollum"	"Erdollum"
6	"тебе пиздец"	"тебе пиздец"
7	"Baltimore Online"	"Baltimore Online"
8	"World Of Hashtags"	"World Of Hashtags"
9	"Patriot Archive"	"Patriot Archive"
10	"St. Louis Online"	"St. Louis Online"

Figure 6: Final Interrogation

Come è possibile vedere risulta come gli attori più influenti all'interno di questa ricerca siano tutti quanti appartenenti alla categoria degli utenti di tipo *Troll*. Questo evidenzia una annosa situazione che ormai da tempo affligge la sfera dei *social network* e non solo: la presenza di utenti fasulli che spesso, come nel caso delle elezioni, si figurano come agenti del caos.

2.0.2 Quali sono le comunità più numerose? Che temi trattano?

Si immagini di voler indagare riguardo alle comunità che sono presenti all'interno del grafo. Per strutturare una esplorazione in tal senso è utile partire da una proiezione, in questo caso di tipo `native`, di alcuni nodi del grafo:

- User
- Troll
- Tweet
- Hashtag

```
CALL gds.graph.project('native-twitter', ['User', 'Troll',
'Tweet', 'Hashtag'], '*')
YIELD graphName, nodeCount, relationshipCount;
```

Partendo dalla proiezione si può sfruttare l'algoritmo di *Louvain* per andare a definire quali siano le comunità all'interno della proiezione che abbiano più di 10 componenti. Il risultato della ricerca viene infine ordinato in maniera non crescente:

```
CALL gds.louvain.stream('native-twitter')
YIELD nodeId, communityId
WITH communityId AS communityId, size(collect(nodeId)) as size
WHERE size > 10
RETURN communityId, size
ORDER BY size DESC;
```

Il metodo Louvain consente di partizionare una rete ottimizzando la modularità . La modularità è un valore compreso tra -1 e 1 che misura la densità dei bordi all'interno delle comunità rispetto a quella dei bordi che collegano le comunità tra loro. L'ottimizzazione della modularità porta teoricamente al miglior partizionamento possibile, tuttavia il suo calcolo effettivo per ciascuno dei possibili raggruppamenti di nodi è troppo lungo nella pratica, da qui l'uso di algoritmi euristici . La prima fase del metodo Louvain consiste nel trovare piccole comunità mediante l'ottimizzazione locale della modularità su ciascuno dei nodi. In secondo luogo, i nodi della stessa comunità vengono raggruppati in un unico nodo e la prima fase viene ripetuta sulla rete appena ottenuta.

"communityId"	"size"
185116	32008
191931	15418
182085	12690
243369	11331
192989	6593
179583	6311
258410	4725
193182	4644
192492	4512
193606	3042

Figure 7: First Interrogation

Risulta possibile approfondire l'interrogazione andando a considerare solamente i gruppi da più di 10.000 utenti e i 5 *hashtag* più utilizzati da ciascuno di essi:

```
CALL gds.louvain.stream('native-twitter', {maxIterations:10})
YIELD nodeId, communityId
WITH communityId AS community, size(collect(nodeId)) as size,
collect(nodeId) as ids
WHERE size > 10000
MATCH (u:User)-->(t)-[*1..2]->(n)
WHERE id(u) in ids
AND (t:Tweet)
AND (n:Hashtag OR n:User)
WITH community as community, size as size,  count(t) as count,
CASE WHEN head(labels(n))='Hashtag' THEN '#'+n.tag
ELSE '@'+n.screen_name END as text
ORDER BY count(t) DESC
RETURN community, size, collect(text)[0..5] AS hashtags_or_users
ORDER BY size DESC LIMIT 10;
```

"community"	"size"	"hashtags_or_users"
234518	37609	["#maga","#trump","#tcot","#trump2016","#neverhillary"]
232686	16465	["#maga","#trump","#tcot","#neverhillary","#pjnet"]
252823	13238	["#rejecteddebatetopics","#betteralternativetodebates","#survivalguidetothanksgiving","#thingsyoucantignore","#ihavearighttoknow"]

Figure 8: Final Interrogation

Viene considerato un massimo di 10 iterazioni dell'algoritmo poiché la sandbox ha risorse computazionali limitati. Qualora si inserisse un valore superiore si avrebbero errori dovuti al fatto che la risposta attenderebbe così tanto ad arrivare, che verrebbe terminata anzi tempo dal server.

Come nel caso della precedente *research question* di seguito una stima delle risorse utilizzate dall'algoritmo utilizzato in questa *search question*:

```
CALL gds.louvain.mutate.estimate('native-twitter',
{mutateProperty:'commnuityID'})
YIELD requiredMemory, bytesMin, bytesMax, heapPercentageMin,
heapPercentageMax, nodeCount, relationshipCount;
```

"requiredMemory"	"bytesMin"	"bytesMax"	"heapPercentageMin"	"heapPercentageMax"	"nodeCount"	"relationshipCount"
"[16240 KiB ... 59 MiB]"	16629889	62146736	0.1	0.1	259253	403943

Figure 9: Resource Utilization

In conclusione ecco l'implementazione della funzione `mutate` per aggiungere il valore della *community* alla proiezione:

```
informazione salvata sul named graph
CALL gds.louvain.mutate('native-twitter', {mutateProperty:'community'})
```

I risultati mostrano come le maggiori comunità siano fortemente a favore del candidato Donald Trump ed allo stesso tempo contrari nella maniera più assoluta a Hillary Clinton, tanto che tra i maggiori *hashtag* troviamo: *#neverhillary*. In aggiunta nella terza comunità più numerose si intravede lo spettro delle *fake news* che hanno flagellato la candidata Hillary, dando adito ad atteggiamenti complottisti con *hashtags* come:

- #thingsyoucantignore
- #ihaverighttoknow
- #tcot

- #rejecteddebatetopics

3 Condiderazioni Finali

Neo4j permette di poter implementare interrogazioni complesse in maniera totalmente *NoSQL* e si rivela essere quindi utilissimo per lo studio di dati strutturati a grafo, soprattutto grazie alla libreria GDS, che mette a disposizione molti algoritmi con cui condurre analisi sempre più approfondite. Rispetto ai risultati ottenuti si conferma quella che era una situazione nota rispetto alle elezioni del 2016: i Troll (pilotati dai russi) hanno fatto da padrone del social network Twitter, seminando il caos e disinformazione, dando adito alla campagna elettorale che ha portato Trump alla vittoria.

References

[1] Manuale Neo4j V5, https://neo4j.com/docs/cypher-manual/current/